

# Software Agents for Efficient Web Server Performance Management

Shadan SANIEPOUR E.<sup>†</sup> and Behrouz Homayoun FAR<sup>†</sup>, *Regular Members*

**SUMMARY** Network traffic characteristics impacts directly network performance, and resource allocation policies. In this work, we introduce a multi-agent system, that manages the performance of web servers with minimal cost of mirroring. In our proposed system each web server is viewed as a software agent that perceives its environment by monitoring its traffic. The goal of the agent is to manage the performance, using cooperative mirror servers, while minimizing the cost of mirroring. Communication between the agents enables each web server to decide about its future actions, which is whether to share its load with the cooperative mirror servers, and how much load to assign to them. The architecture of a software agent that is intended to manage the performance of a web server, is elaborated and its different modules are described. Also a set of cooperative agents is defined, that form a multi-agent system and is intended to assure maintaining the performance with minimal cost of mirroring. The experimental results presented in this article illustrates the effectiveness of the proposed system.

**key words:** *software agent, performance, mirroring, queuing theory, optimization*

## 1. Introduction

The bursty and unpredictable nature of Web traffic complicates managing performance of the web servers. To maintain the performance of a web server allocating sufficient resources (i.e., memory, bandwidth, etc.) is crucial. To this end, mirror servers are a way of providing auxiliary resources. In order to guarantee the performance of a web server, one solution is to allocate as much mirror servers to handle the load even under the heaviest traffic. Obviously this solution is costly, and as the web traffic grows exponentially, it turns inefficient.

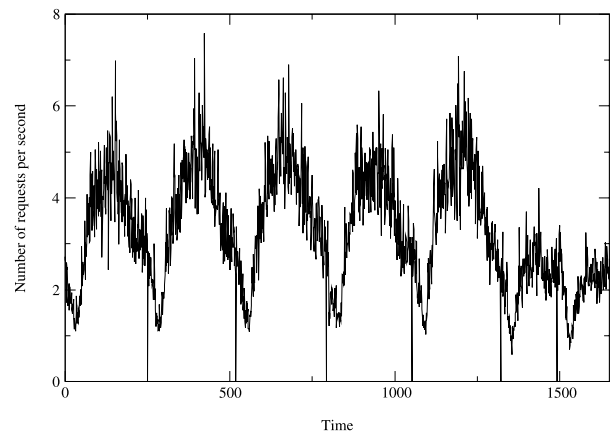
Recent studies on traffic issues related to web servers have shown distinct patterns in the amount of traffic on a web server depending on the time of a day: peak time during busy hours and the time that web server is lightly loaded [2], [6], [15]. Figure 1 shows the traffic of a typical busy server in terms of requests per second extracted from [4]. This phenomenon suggests the possibility of using mirroring services only during peak time and thus provides a means of sharing mirror servers in order to reduce the cost.

In this article we introduce a multi-agent system,

Manuscript received June 25, 2001.

Manuscript revised November 6, 2001.

<sup>†</sup>The authors are with the Faculty of Information and Computer Science, Saitama University, Saitama-shi, 338-8570 Japan.



**Fig. 1** Traces of traffic of a busy server (Clarknet server) for duration of one week.

that manages the performance of web servers. A web server is viewed as software agent. A set of cooperative mirror servers are proposed as cooperative agents and a cost is associated with the mirrored load to prevent the servers against assigning infinite load to their mirroring partners. Performance is evaluated by means of latency. If a server fails to provide a reasonable expected latency most of the clients are likely to abandon retrieving data. This situation is referred to as performance deficiency and a server that demonstrates performance deficiency is referred to as an overloaded server. The notion of partial and temporal mirroring, introduced in this article, allows applying mirroring only when the server is overloaded, and also to transfer as much load to maintain the performance. The main contribution of this work is to assure the performance of web servers with minimum cost of mirroring.

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment to achieve its goals [16]. The main characteristic of an agent is to be autonomous: act independently and without human intervention, goal oriented, and flexible: account for the changes in the environment. This framework distinguishes an agent from an ordinary software component. Also it makes it easier to divide the functionality of an agent to number of tasks. Each task can then be handled by a software module. The architecture of the individual agents as well as the relation between different agents is il-

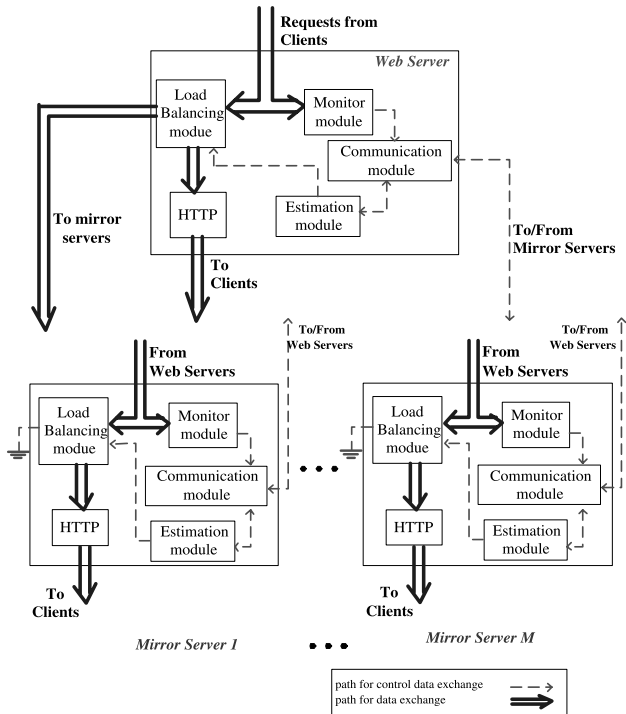


Fig. 2 Software modules for managing performance of web servers.

illustrated in Fig. 2. The functionality of each agent is divided to number of tasks where each task is carried out by a software module. Viewing the web server as an agent, enables us to clearly distinguish the responsibility of each module. The *Monitor module* is responsible to perceive the agents environment. It models a web server as a queuing system and monitors its performance. In case of detecting performance deficiency, *Communication module* enquires the traffic situation of mirror servers, and according to this information *Estimation module* decides about its future action by estimating the optimal mirroring solution, which is, deriving the best cooperative mirror servers at any time, and also the mirroring contents. In fact by communicating with the other agents, an agent will be able to account for the changes in the environment. The *Load balancing module* is responsible for distributing load among cooperative mirror servers. The *estimation module* in the mirror server side is responsible to ensure that cooperating with an overloaded server to handle the decided portion of the load does not violate the performance of the mirror server. The *load balancing module* in the mirror server side, drops some of the requests, if the load delegated by the overloaded server exceeds the previously agreed amount of load and causes performance deficiency in the mirror server side.

The rest of this paper is organized as follows. In the next section we study the issues related to performance of web servers and propose a queuing system to model a web server. According to this model we de-

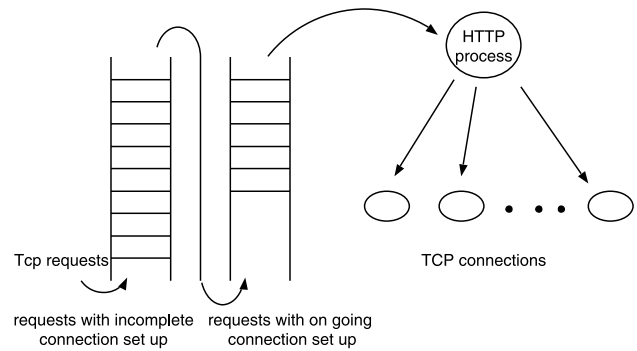


Fig. 3 HTTP receives the requests queued by TCP, and serve them via multiple TCP connections.

rive the parameters that should be monitored by the *monitor module*. In Sect. 3 we analyze the parameters that their change impact the performance. These parameters should be enquired from all the mirror servers via *communication module* to let the *estimation module* estimate the performance after load sharing. Section 4 describes how the *estimation module* estimates the performance of an overloaded server if a portion of load is lifted from it, and also the performance of a mirror server if an extra amount of load is assigned to it. Consequently we give an analytical solution for the *estimation module* to find an optimal mirroring solution. Section 5 elaborates on how the *load balancing module* distributes the load among the mirror servers. Experimental results are presented in Sect. 6. Finally we conclude in Sect. 7.

## 2. Monitor Module

Monitor module is responsible to detect the performance deficiency. In this section we describe the parameters that are required to be monitored by the *monitor module*, to allow detecting performance deficiency.

A web server uses HTTP protocol implemented over TCP/IP, for data transfer. Therefore analyzing the performance of a web server is correlated with the behavior of TCP. Figure 3 illustrates how requests are handled by the HTTP process. As illustrated in this figure, TCP process buffers all the incoming requests with incomplete connection setup. A separated buffer is allocated for the requests with still ongoing connection setup. HTTP process establishes a TCP connection to serve each request. Concurrent web servers allow establishing simultaneous TCP connections each serving an HTTP request. The lifetime of each connection is correlated with the throughput of the TCP process and the size of the requested document.

As the requests are queued before processing, the behavior of a web server can be described as a queuing system. Expected latency can be estimated according to this model. Similar formulation is proposed to model a multiprocessor environment [10], [19], where the goal

is to maintain the shortest queue for all the processors. We are however interested in the average latency of the servers.

Recent studies on modeling the distribution of inter-arrival times of the requests and service time of web servers have shown that traditional exponential distribution are inadequate to model the behavior of web traffic [6], [7], [11]. We consider a general distribution for inter-arrivals and also general distribution for service time. Since the buffer size is relatively large<sup>†</sup>, we approximate the web servers as queuing system with infinite buffer size. Consequently our queuing model follows a general form of  $GI/G/c$ , where  $c$  denotes the total number of servers (i.e., maximum number of simultaneous connections created by the web server).

If the inter-arrival time of the requests is denoted as  $a$  and service time is denoted as  $s$ , the arrival rate is given by  $\lambda = 1/E(a)$  and service rate is given by  $\mu = 1/E(s)$ . The utilization  $\rho = \frac{\lambda}{c\mu}$ , is known as the mean fraction of active servers [1]. In practice  $\lambda$  increases gradually when getting closer to the peak time, and decreases gradually around less busier hours (Fig. 1). Therefore the *monitor module* should dynamically monitor the mean and variance of inter-arrivals and also service time, in order to keep trace of changes in the data. These are the parameters that allow estimating the latency. The expected latency of a queuing system can be derived as:

$$W = W_Q + 1/\mu \quad (1)$$

where  $W_Q$  denotes the waiting time in the queue, and according to [1] can be estimated as:

$$W_Q = \frac{\rho/\mu}{1-\rho} \frac{cv_a^{f(cv_a, cv_s, \rho)} + cv_s^2}{2} \rho^{(c-1)/4} \quad (2)$$

$$f(cv_a, cv_s, \rho) = \begin{cases} 1 & cv_a \in \{0, 1\} \\ (\rho(14.1cv_a - 5.9) + (-13.7cv_a + 4.1))cv_s^2 \\ \quad + (\rho(-59.7cv_a + 21.1) + (54.9 - 16.3))cv_s \\ \quad + (\rho(cv_a - 4.5) + (-1.5cv_a + 6.55)) & 0 < cv_a < 1 \\ -0.75\rho + 2.775 & cv_a > 1 \end{cases} \quad (3)$$

$cv_a$  denotes the coefficient of variation of inter-arrival time, and  $cv_s$  denotes the coefficient of variation of service time.

Our goal is to keep the expected latency of the system under a desirable threshold denoted as  $T_0$ . *Monitor module* monitors the latency, as well as mean and variance of inter-arrival time and also service time.

In the next section we describe how transferring the load can tune the parameters of (2), and also derive the parameters that must be reported to the *estimation module* by the *communication module* to let estimating

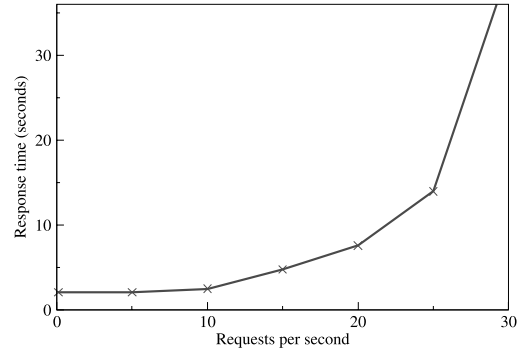


Fig. 4 Changes in latency when arrival rate increases.

the latency after load sharing.

### 3. Communication Module

In this section we study the parameters that influence the latency. These are the parameters that are reported by *communication module* to let the *estimation module* estimate the latency after tuning the performance. *Communication module* extracts these parameters from its peer modules in the mirror servers side.

According to (2), latency is a function of  $\rho = \frac{\lambda}{c\mu}$ ,  $\mu$ ,  $cv_s$  and  $cv_a$ . To reduce the latency we can increase the  $\mu$ , and/or decrease  $\lambda$ ,  $cv_s$ ,  $cv_a$ .

$\lambda$  can be reduced by using a switch (e.g., L7 switch) that transfers some of the requests to a mirror server. Figure 4 illustrates how latency can be tuned if  $\lambda$  is reduced.

Using a switch to transfer some of the requests results in having longer inter-arrival times. In order to estimate the coefficient of variation of the arrival rate after transferring a portion of the load, we need to decompose the arrival stream to its corresponding streams of requests for different documents. Given that we know the rate of changes in the request rate of different documents we can estimate the total change in the coefficient of variation of the arrival rate. If the inter-arrival time of the requests corresponding to different documents are batch-Poisson and Batch-deterministic process with geometric batch sizes, the total coefficient of variation of the inter-arrival times according to (8) of [21] can be derived from (4). Note that assumption of geometric batch sizes makes the individual clients inter-arrival time independent and identically distributed (GI process) [21].

$$cv_a^2 = \sum_{i=1}^K cv_i^2 \frac{n_i^{new}}{N}$$

<sup>†</sup>In Solaris by default, maximum buffer size for connections with incomplete setup is (`tcp_conn_req_max_q0 = 1024`), and Maximum buffer size for the connection with ongoing connection setup is (`tcp_conn_req_max_q = 128`).

$$= \sum_{i=1}^K cv_i^2 \frac{\hat{q}_i^{new}}{\lambda} \quad (4)$$

where  $cv_i$  denotes the coefficient of variation for the inter-arrival time of the requests for document  $i$  and  $n_i^{new}$  and  $\hat{q}_i^{new}$  respectively denote the number of requests corresponding to document  $i$  and the request rate corresponding to document  $i$ .

To alter the  $\mu$  we show that  $\mu$  has an inverse proportion with average transferred document size. Therefore by altering average document size we can tune the  $\mu$  and consequently  $\rho$  and ultimately the latency.

If the connection throughput of each TCP connection is denoted as  $CT$ , the service time of a TCP connection  $s$  to serve the document  $D$  satisfies:

$$D = \int_0^s CT(t)dt \quad (5)$$

$$\begin{aligned} E(D) &= E\left(\int_0^s CT(t)dt\right) \\ &= E_s\left(E\left(\int_0^x CT(t)dt|s=x\right)\right) \\ &= E_s(sE(CT)) \\ &= E(s)E(CT) \end{aligned}$$

and finally we will have:

$$\mu = \frac{1}{E(s)} = \frac{E(CT)}{\bar{D}} \quad (6)$$

$\bar{D}$  indicates the average document size, in KB. Note that  $CT$  is assumed to be stationary. Equation(6) implies that  $\mu$  can be tuned by adjusting  $\bar{D}$  or  $CT$  independently. However, it has been shown that  $CT$  is a function of  $RTT$ , packet loss and other limitations imposed by the client (maximum window size) and adjusting it is not under our control [3], [9], [12]. Our objective is to tune the  $\mu$  by altering  $\bar{D}$ . Following equation shows how  $\mu$  is changed if we alter average document size.

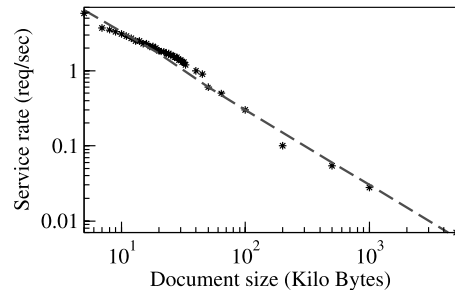
$$\mu_{new} = \mu \cdot \frac{\bar{D}}{\bar{D}_{new}} \quad (7)$$

where  $\mu_{new}$  and  $\bar{D}_{new}$  denote the values after change. The verification of this equation is illustrated in Fig. 5.

The average document size is given by:

$$\bar{D} = \frac{\sum_i n_i d_i}{N} \quad (8)$$

where  $N$  denotes total number of requests received at this server,  $n_i$  denotes the number of requests corresponding to document  $i$ , and  $d_i$  denotes the size of document  $i$  in kilo bytes. Considering the fact that  $n_i = \hat{q}_i \cdot t$  where  $\hat{q}_i$  denotes the total request rate corresponding to document  $i$ , and  $N = \lambda \cdot t$ , Eq.(8) is rewritten in (9) to indicate the relation between  $\bar{D}$  and request rate. Note that  $t$  refers to the sampling period



**Fig. 5** Changes in service rate when average retrieved document size increases, plotted in a logarithmic scale.

that results in having sufficient number of samples.

$$\bar{D} = \frac{\sum_i \hat{q}_i d_i}{\lambda} \quad (9)$$

According to (9) reducing average document size is possible via reducing the request rate corresponding to some of the documents, which is possible by transferring part of the requests to a mirror server.

With the similar reasoning as we showed for  $\mu$  we can show that the coefficient of variation for service time can be derived from the following equation.

$$(cv_s^{new})^{new} = \frac{\bar{D}^{2new}/(\bar{D}^{new})^2}{\bar{D}^2/(\bar{D})^2} (cv_s^2) - 1 \quad (10)$$

where  $\bar{D}^{new}$  is derived from:

$$\bar{D}^{new} = \frac{\sum_i \hat{q}_i d_i^2}{\lambda} \quad (11)$$

In this section we studied the impact of load transfer on tuning  $\lambda$ ,  $\bar{D}$ ,  $\mu$ ,  $\bar{D}^2$  and  $cv_a$ . These parameters must be enquired from each mirror server via *communication module* to let the *estimation module* estimate the latency after applying load sharing.

In the next section we describe how the *estimation module* can estimate the latency after load sharing.

#### 4. Estimation Module

Transferring a portion of the load to a mirror server, alters the performance of both parties, the server that transfers its load and the mirror server that receives extra load. In this section we describe how the *estimation module* estimates the latency for the overloaded server and mirror servers after transferring the load. To clarify the terminology, subscript  $j$  is used to specify the parameters of mirror servers, whereas subscript  $o$  specifies the parameters of the overloaded server. The integer  $M$  denotes the number of mirror servers, and the integer  $K$  denotes the number of documents on the overloaded server. The request rate corresponding to document  $i$  which is assigned to the mirror server  $j$  is denoted as  $q_{ij}$ , and the total request rate corresponding to document  $i$  is denoted as  $\hat{q}_i$ .

For an overloaded server, the new value of  $\lambda$  after load sharing, depends on the applied load transfer method. If load is transferred to the mirror server via redirection  $\lambda$  remains unchanged as the requests arrive at the overloaded server and then get redirected. The average document size in this case is derived from (12).

$$\begin{aligned}\overline{D}_o^{new} &= \frac{\sum_{i=1}^K \hat{q}_i d_i - \sum_{i=1}^K \sum_{j=1}^M \hat{q}_{ij} d_i}{\lambda_o} \\ &= \overline{D}_o \left(1 - \frac{\sum_{i=1}^K \sum_{j=1}^M \hat{q}_{ij} d_i}{\lambda_o}\right)\end{aligned}\quad (12)$$

If however we use a switch (e.g., L7 switch) to transfer the requests to the mirror servers before they get to the overloaded server, the value of  $\lambda$  will be reduced by the amount of the request rate that is transferred to the mirror servers. The new  $\lambda$  in this case is derived by:

$$\lambda_o^{new} = \lambda_o - \sum_{i=1}^K \sum_{j=1}^M q_{ij} \quad (13)$$

The average document size for the case of switching is given by:

$$\overline{D}_o^{new} = \frac{\lambda_o \overline{D}_o - \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i}{\lambda_o - \sum_{i=1}^K \sum_{j=1}^M q_{ij}} \quad (14)$$

$\mu_o^{new}$  is derived from (7) and therefore we can derive  $\rho_o^{new}$ . Finally to be able to estimate the new value of latency, new values of coefficient of variation for inter-arrival time and the service time should be estimated. Coefficient of variation of the inter-arrival times for the case of redirection remains unchanged and for the case of switching can be estimated according to (4). To derive the coefficient of variation for the service time using (10) we need to estimate  $\overline{D}^2^{new}$ :

$$\overline{D}^2^{new} = \begin{cases} \frac{\lambda_o \overline{D}_o^2 - \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i^2}{\lambda_o} & \text{redirection} \\ \frac{\lambda_o \overline{D}_o^2 - \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i^2}{\lambda_o - \sum_{i=1}^K \sum_{j=1}^M q_{ij}} & \text{switching} \end{cases} \quad (15)$$

Next is to estimate the new performance parameters for each mirror server. The value of  $\lambda$  for each mirror server increases as the transferred request rate increases by the overloaded servers. The new value of  $\lambda$  for each mirror server is given by (16). The average document size is derived from (17).

$$\lambda_j^{new} = \lambda_j + \sum_{i=1}^K q_{ij} \quad (16)$$

$$\overline{D}_j^{new} = \frac{\lambda_j \overline{D}_j + \sum_{i=1}^K q_{ij} d_i}{\lambda_j + \sum_{i=1}^K q_{ij}} \quad (17)$$

$\mu_j^{new}$  is derived by (7) and therefore we can derive

$\rho_j^{new}$ . The coefficient of variation for inter-arrival times is estimated by (18).

$$(cv_{aj}^2)^{new} = \frac{cv_{aj}^2 \lambda_j + \sum_{j=1}^M \sum_{i=1}^K cv_i^2 q_{ij}}{\lambda_j + \sum_{i=1}^K q_{ij}} \quad (18)$$

The coefficient of variation of the service time  $cv_s$  is derived from (10).  $\overline{D}^2^{new}$  for each mirror server is given by:

$$\overline{D}^2^{new} = \frac{\lambda_j \overline{D}_j^2 + \sum_{i=1}^K q_{ij} d_i^2}{\lambda_j + \sum_{i=1}^K q_{ij}} \quad (19)$$

Finally substituting the new values of  $\rho$ ,  $\mu$ ,  $cv_s$  and  $cv_a$  in (1) and (2) enables the *estimation module* to estimate the latency of overloaded server ( $W_o^{new}$ ) and also each mirror server ( $W_j^{new}$ ) after load sharing. In the case of using redirection as the load transfer method, the actual waiting time perceived by the client is more than what is estimated by the above equations. From client viewpoint, all the requests arrive at the overloaded server and wait in the queue until their turn to be processed, but after that some of them are processed and the others are redirected to the mirror servers and wait in the new queue and are processed with different service time. Equation(20) formulates this waiting time.

$$\begin{aligned}W_o^{client} &= W_{Q_o}^{new} + \frac{\sum_{j=1}^M n_j W_j^{new} + (N_o - \sum_{j=1}^M n_j) \frac{1}{\mu_o^{new}}}{N_o} \\ &= W_{Q_o}^{new} + \frac{1}{\mu_o^{new}} + \sum_{j=1}^M \frac{\sum_{i=1}^K q_{ij}}{\lambda_o} W_j^{new} - \sum_{j=1}^M \frac{n_j}{N_o} \frac{1}{\mu_o^{new}} \\ &= W_o^{new} + \sum_{j=1}^M \frac{\sum_{i=1}^K q_{ij}}{\lambda_o} W_j^{new} - \sum_{j=1}^M \frac{\sum_{i=1}^K q_{ij}}{\lambda_o \mu_o^{new}}\end{aligned}\quad (20)$$

$n_j = \sum_{i=1}^K q_{ij} \cdot t$  denotes the total number of requests transferred to the mirror server  $j$ , and  $N_o = \lambda_o \cdot t$  the total number of requests arrived at overloaded server within  $t$  seconds.

Our motivation is to transfer minimal load to mirror servers, in order to receive the minimal charge, while keeping the latency of the server under a tolerable threshold. We formulate this problem as an optimization problem with inequality constraints. The constraints are: to keep the latency of the overloaded server and all the cooperating mirror servers under the threshold. A mirror server that gets overloaded is not a choice for improving the performance. The total load transferred to mirror servers is given by:

$$transferred\_load = t \cdot \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i \quad (21)$$

where  $t$  denotes the period of time the server is overloaded and mirroring continues. The optimization

problem is formulated as follows:

$$\min \sum_{i=1}^K \sum_{j=1}^M q_{ij} d_i \quad (22)$$

s.t.

$$W_o^{client} - T_0 \leq 0 \quad (23)$$

$$W_j^{new} - T_0 \leq 0 \quad j = 1, 2, \dots, M \quad (24)$$

$$\rho_j \left( 1 + \frac{\sum_{i=1}^K q_{ij} d_i}{\lambda_j D_j} \right) - 1 < 0 \quad j = 1, 2, \dots, M \quad (25)$$

$$\left( \sum_{j=1}^M q_{ij} \right) - \hat{q}_i \leq 0 \quad i = 1, 2, \dots, K \quad (26)$$

$$q_{ij} \geq 0 \quad i = 1, 2, \dots, K \\ j = 1, 2, \dots, M \quad (27)$$

$\hat{q}_i$  is the total arrival rate for document  $i$  at the overloaded server. The task is to determine the value  $q_{ij}$  for  $i = \{1, 2, \dots, K\}$  and  $j = \{1, 2, \dots, M\}$ , which minimizes the transferred load under above constraints. As we see some of the constraints are nonlinear. The general procedure to deal with this type of problems, is to generate a sequence of convex, explicit subproblems and solve them in an iterative fashion, which is called Sequential Convex Programming (SCP) [8]. Various approximation schemes have been developed for this purpose. We used the globally convergent Method of Moving Asymptotes (MMA) [17] as defined in [18]). To solve the subproblems generated by this algorithm, we have used Lagrangian relaxation problem, solved by conjugate subgradient method. As the result, an optimal answer for  $q_{ij}$ ,  $j = \{1, 2, \dots, M\}$ ,  $i \in \{1, 2, \dots, K\}$  is derived.

In this section we described how the *estimation module* estimates the performance after load sharing, and how it decides about the optimal mirroring solution. In the next section we describe how *load balancing module* uses this information to distribute the load among the mirror servers.

## 5. Load Balancing Module

After deriving the optimal request rate that should be assigned to each mirror server, a copy of the documents is sent to their selected mirror servers. *Load balancing module* is responsible for distributing the incoming traffic among the cooperating mirror servers.

To distribute the load among mirror servers, in order to achieve the desired  $q_{ij}$ , we first derive the proportion of load related to each document which is intended to transfer to each mirror server. This is derived from the following equation.

$$p_{ij} = q_{ij} / \hat{q}_i \quad (28)$$

The *load balancing module* assigns any request to

the document  $di$ , to the mirror server  $j$ , with the probability  $p_{ij}$ . As time passes and load forwarding continues, the request rate corresponding to document  $di$  assigned to the mirror server  $j$  converges to  $q_{ij}$ . As the conclusion, the problem of traffic routing which is an issue in mirroring, is reduced to distribute the extra amount of load with the derived probability.

## 6. Experimental Results

The goal of this experiment is to show:

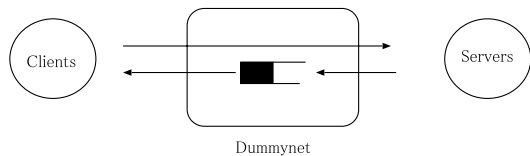
- Mirroring is necessary only if we have a heavy load that causes performance deficiency, to illustrate the feasibility of temporary mirroring.
- Partial mirroring can achieve a reasonable performance. We then compare the cost of mirroring in terms of the transferred load for the case of full mirroring and partial mirroring, and show that partial mirroring reduces the cost, while assuring to maintain the performance.

Note that full mirroring refers to conventional mirroring, where the load is equally distributed among mirror servers. Consequently probability of assigning a request corresponding to document  $i$  to mirror server  $j$  is  $\frac{1}{M+1}$ . Partial mirroring in the other hand refers to our approach, where probability of assigning each request to each mirror server is derived from (28). Interestingly as we see later, partial mirroring guarantees maintaining the performance under heavy load even if full mirroring fails to maintain the performance for the same load.

In the following we first describe the test environment and the measurements.

### 6.1 Test Environment

In our experiment, we use a group of three servers, running Apache software (version 1.3.19), each running on a Solaris 2.8 operating system. One of the servers is intended to simulate the behavior of web servers and the other two are mirror servers. We use a traffic generator benchmark, which will be described in detail later, to generate a heavy load on the server. Since the experiment is applied in a LAN, we use *dummynet* [14], as the network emulator, to simulate the characteristic of a WAN. Figure 6 shows this simulation. We define a multipath channel between the benchmark and the web server to account for a variety of possible paths. In particular we create four pipes, with different probabilities, each representing a class of traffic on a WAN. The bandwidth for these pipes are: 256 Kb, 400 Kb, 600 Kb and 1 Mb. Delay in terms of msec defined by RTT (Round Trip Time) is defined as 80 msec. Note that as we see in Fig. 6 the requests sent to the server are not queued. This is because we tune the request rate in our Benchmark so that generates a simulated

**Fig. 6** Test environment.**Table 1** Workload generated by the benchmark.

File size (KB)	Probability of access
0.5	0.35
5	0.5
50	0.14
500	0.009
5000	0.001

inter-arrival rate on the server. However, when serving the requests, packets are queued before getting to client to simulate the WAN bandwidth and latency.

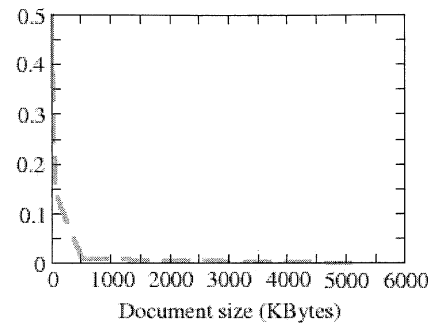
## 6.2 Traffic Generator

Our traffic generator benchmark written in C runs on FreeBSD operating system. The workload generated by this benchmark is intended to simulate the real workload of web servers. Work reported in [5] approves that file size distribution of the WWW has a heavy tailed distribution: most documents are small (a few kilo bytes) but the number which are very long tend to contribute the majority of traffic. The file set that simulate this characteristics for our benchmark is adopted from webstone benchmark [20], and the specification is given in Table 1.

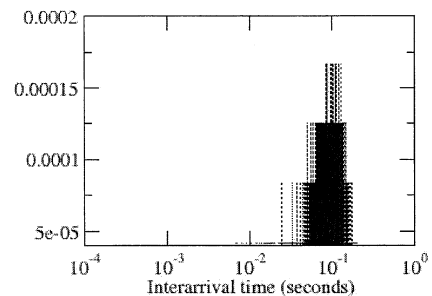
Inter-arrival time of WWW requests follow a Weibull distribution [7]. This is due to the burstiness nature of inter-arrival time of the requests. Applications such as Web, create inter-arrival times from very short to very long. Very short inter-arrival times are mainly due to the embedded objects within a homepage (e.g., images). Currently these embedded objects are retrieved via parallel TCP connections, limited to four connections. Use of persistent connections [13], reduces this number to two, but still leaves the possibility of small inter-arrivals. Moreover, Weibull distribution has the advantage to include exponential distribution, which traditionally has been used to model the arrivals, as a special case. Mathematically the distribution is defined as:

$$f(x) = \frac{\beta x^{\beta-1}}{\eta^\beta} e^{-(\frac{x}{\eta})^\beta} \quad \beta, \eta > 0, x \geq 0 \quad (29)$$

where  $\beta$  is shape parameter defining the shape of the distribution and  $\eta$  is scale parameter.  $\beta > 3.06$  creates a left heavy tailed distribution, which matches the characteristics of inter-arrivals [7]. The mean of this distribution is given by:



(a)



(b)

**Fig. 7** Benchmark characteristics: (a) probability density function for document size. (b) probability density function for inter-arrival time for  $(\lambda = 10)$  req/sec.

$$E(x) = \eta \Gamma\left(1 + \frac{1}{\beta}\right) \quad (30)$$

In our benchmark  $\beta = 5$  simulates the left heavy tail nature of inter-arrivals, and  $\eta$  is assigned a value to simulate the intended request rate ( $\lambda = 1/E(a)$ ). In fact in our experiment,  $\eta$  is altered, to gradually increase the request rate.

The probability density function of inter-arrival time and document size generated by this benchmark is shown in Fig. 7.

This benchmark generates simultaneous clients each running as separate threads. The waiting time is measured, as perceived by the client. Each client thread measures the waiting time from when it sends the request until when it receives the last byte. In the case of redirection, it measures the time from when it sends the request to the original server until when it receives the last byte of the requested document from the mirror server. The benchmark reports the average waiting time, as perceived by the clients.

## 6.3 Measurements

In order to measure the inter-arrival statistics, we implemented a packet capture program to collect all the incoming packets to the port 80 (HTTP port) which have their SYN flag set. These packets represent HTTP incoming requests. To measure the mean of inter-arrival time, we measure the time between each two consequent requests, and update the mean and vari-

ance of the inter-arrival times sequentially.

Measuring service time is a little more complicated. The response time of a web server refers to the time that takes for the HTTP process to send the whole document to the client. Therefore we measure the time from when the HTTP process starts processing request, until when the last FIN is acknowledged, indicating the end of data transfer. We modified Apache to assign a time stamp to each connection as soon as it accepts a new request. To find out when a connection is closed, we capture all the incoming and outgoing packets from port 80, that have their FIN flag set. As soon as the last FIN on a connection is acknowledged the connection is considered closed and the time between accepting the request and closing the connection is reported as service time.

The maximum number of connections in our Apache server is set to 50 connections. This defines the number of servers ( $c$ ) in estimating waiting time.

#### 6.4 Results

We use our web traffic generator benchmark and gradually increase the load of our web server. Changes in the traffic are applied every 30 minutes. We define the threshold of 5 seconds as an upper limit of tolerable latency. Each experiment is repeated for the case that load is redirected to the mirror servers, and the case that load transfer is applied via switching. We have simulated the switching in the client side (in the benchmark), and our goal is to show if requests are transferred to the mirror server, via a switch, before getting to the overloaded server, performance can be improved.

Figure 8 compares the latency with and without using partial and temporal mirroring, when load is redirected to the mirror servers, and when switching has been used to transfer the load. Waiting time is what is reported by the benchmark and represents what actually the client observes. Note that in case of full mirroring the probability of assigning a request to mirror servers is simply  $M \cdot \frac{1}{M+1}$ , where  $M = 2$  and therefore probability of assigning a request to mirror servers is  $\frac{2}{3}$ . In the other hand, this probability for case of partial mirroring in our approach, is derived from (28) and is shown in Figs. 10(a) and (b). As we see in Fig. 8, to provide an 5 seconds latency, mirroring is required only when the number of incoming requests per second exceed 20 (around 2 million hits per day). This request rate corresponds to the peak time. In fact we need to use mirroring services only in busy hours. Also, as illustrated in this figure, partial mirroring with two mirror servers allows maintaining a 5 seconds latency for over 5.6 million hits per day (65 requests per second), while applying full mirroring with one mirror server can maintain this latency only for up to 35 requests per second. Full mirroring with two mirror servers increase this number to almost 50 requests per second, which is

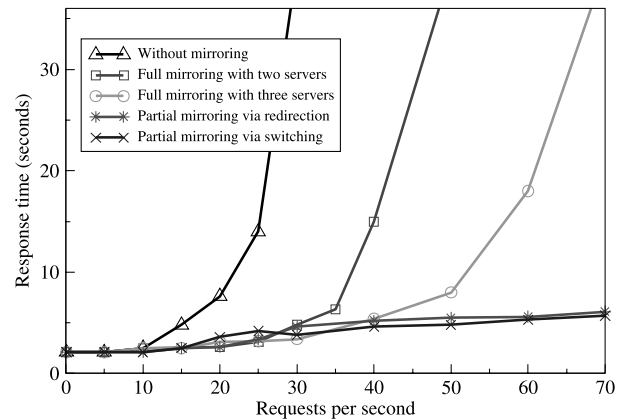


Fig. 8 Comparing latency with and without applying partial mirroring.

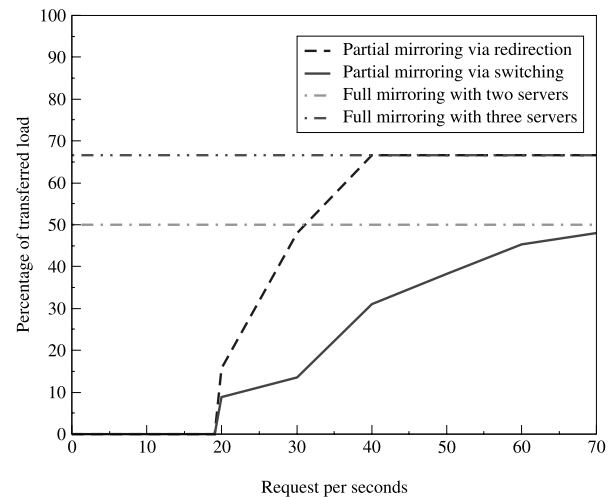


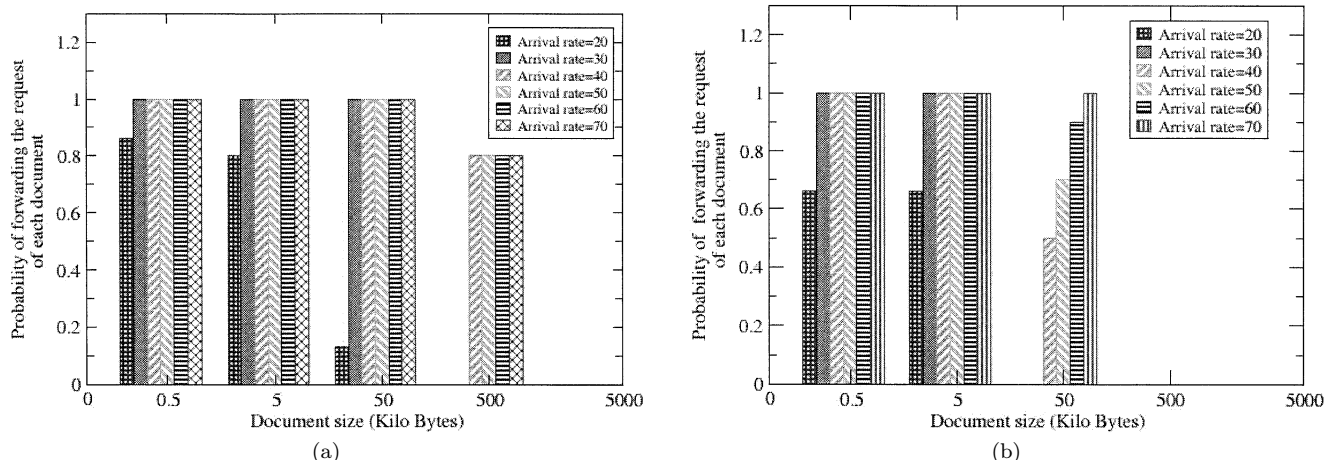
Fig. 9 Comparing the amount of transferred load to the mirror servers for the case of full mirroring, partial mirroring via redirection, and partial mirroring via switching.

still less than what can be achieved by partial mirroring. The reason is, in full mirroring we simply distribute the load and the request rate equally among the mirror servers, consequently having three servers that receive 50 requests per second for the average document size generated by the benchmark, is almost similar to having one web server that receives 20 requests per second<sup>†</sup>, which according to Fig. 8 fails to provide the 5 second latency. However our optimization algorithm generates different average document sizes on mirror servers and allows us to distribute the request rate and the load, so that we have minimum transferred load while maintaining 5 seconds latency.

Figure 9 compares the cost in terms of transferred load for full mirroring and partial mirroring.

As illustrated in this figure, although using redi-

<sup>†</sup>We expect to handle  $20 \times 3 = 60$  request with three servers, however the overhead of request forwarding decreases this amount to 50.



**Fig. 10** Probability of forwarding the request corresponding to each document to the mirror servers (i.e.,  $\sum_{j=1}^M p_{ij}$ ) (a) via redirection (b) via switching.

rection as the load transfer method, can alleviate the cost in contrast to full mirroring, using a switch is recommended since it can effectively reduce the cost. As we see in this figure, for up to 50 requests per second in contrast to full mirroring with 3 servers, using a switch can reduce the transferred load to half (and if the request rate is less than 40 it is reduced to one third), indicating the fact that mirror servers can be shared among two (to three) overloaded servers in the worst case (i.e., all the servers are overloaded by up to 50 requests per second). For more than 50 requests per second, partial mirroring is the only choice, full mirroring fails to maintain the performance. The main conclusions are as follows:

- Partial mirroring can maintain the desired latency, even under the heavy traffic when full mirroring fails to maintain this latency.
- According to our experiment with set of two mirror servers, partial mirroring can reduce the cost to half (or one third if the arrival rate is less than 40 requests per second). Allocating more mirror servers is expected to give better results.
- This approach allows applying mirroring only during peak time, therefore when the server is lightly loaded there is no need to apply mirroring and mirroring services is given to other overloaded servers.

Finally to derive an insight about how the load is transferred, Fig. 10 (a) illustrates the probability of assigning a request of each document to mirror servers, as derived from (28) in case of redirection, and Fig. 10 (b) illustrates this probability for the case of switching.

As can be concluded from Fig.10(a) and Fig.10(b) in the case of redirection, where the only way to reduce  $\rho$  and therefore latency is to tune the service rate  $\mu$ , it is preferred to transfer the traffic generated by bigger files in contrast to switching where it is preferred to transfer the load generated by the smaller

files.

It is also worth noting that this multi-module system adds only a little overhead to the regular HTTP server. In fact it consumes less than 2 minutes of CPU time per hour.

## 7. Conclusions

In this article we introduced a multi-agent system that manages the performance of web servers with minimal cost of mirroring. The concept of software agent used in this article allows us to clearly define the domain of activity of each agent and its modules. Each web server, viewed as a software agent perceives its environment by monitoring its traffic. In case of detecting performance deficiency, it communicates with other agents in the mirror servers side, to retrieve the necessary information for deciding about its future actions: deciding about the mirroring partner(s), and about the amount of load it assigns to each mirroring partner. The advantage of using multi-agent system and benefiting from their communication is that we can apply dynamic load balancing, which is, at any time we can decide about the mirroring partners and also the mirroring contents. In fact this system enables us to implement the notion of partial and temporal mirroring, and provides a means for reducing the cost of mirroring. Experimental results presented in this paper, confirms the effectiveness of this approach.

## References

- [1] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science*, A Wiley-Interscience Publication, 1998.
- [2] R.B. Bunt and D.L. Eager, "Achieving load balance and effective caching in clustered Web servers," *The 4th International Web Caching Workshop*, San Diego, California, April 1999.

- [3] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," Proc. 2000 IEEE Infocom Conference, Tel Aviv, Israel, March 2000.
- [4] ClarkNet traffic, <http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html>
- [5] M.E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," IEEE/ACM Trans. Networking, vol.5, no.6, pp.835-846, Dec. 1997.
- [6] J. Dilley, "Web server workload characterization," HP Labs Technical Reports HPL-96-160, Dec. 1996.
- [7] A. Fledmann, "Characteristics of TCP connection arrivals," Technical Report, AT&T Labs Research, Dec. 1998.
- [8] C. Fleury, "Sequential convex programming for structural optimization problems," in Optimization of Large Structural Systems, ed., G. Rozvany, volume I of NATO ASI, pp.531-553. Kluwer Academic Publishers, 1993.
- [9] S. Floyd, "Connections with multiple congested gateways in packet switching networks, Part1: One-way traffic," Computer Communications Review, vol.21, no.5, Oct. 1991.
- [10] H. Guyennet, F. Spies, and M. Trehel, "Modeling and simulation of dynamic load balancing using queuing theory," Parallel Algorithms and Applications, Gordon and Breach Science Publishers, vol.5, nos.1-2, 1995.
- [11] B. Liu, "Web traffic latency: Characteristics and implications," J. Universal Computer Science, vol.4, Issue 9, 1998.
- [12] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," SIGCOM'98, Sept. 1998.
- [13] V.N.M. Padmanabhan and J. Mogul, "Improving HTTP latency," Computer Networks and ISDN Systems, vol.28, no.1/2, pp.25-35, Dec. 1995.
- [14] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," ACM Computer Communication Review, vol.27, no.1, pp.31-41, Jan. 1997.
- [15] J.W. Roberts, "Traffic theory and the internet," IEEE Commun. Mag., vol.39, no.1, pp.94-99, Jan. 2001.
- [16] S.J. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, p.33, Printice Hall, Englewood Cliffs, NJ, 1995.
- [17] K. Svanberg, "Method of moving asymptotes — A new method for structural optimization," International J. for Numerical Methods in Engineering, vol.24, pp.359-373, 1987.
- [18] K. Svanberg, "The MMA for modeling and solving optimization problems," Proc. 3rd WCSMO (The World Congress of Structural and Multidisciplinary Optimization), Buffalo, New York, May 1999.
- [19] M. Trehel, C. Balayer, and A. Alloui, "Modeling load balancing inside groups using queuing theory," PDCS'97, 10th International Conference on Parallel and Distributed Computing System, New Orleans, Louisiana, Oct. 1997.
- [20] Webstone, <http://www.mindcraft.com/webstone>
- [21] W. Whitt, "Towards better multi-class parametric-decomposition approximations for open queuing networks," Annals of Operations Research, vol.48, pp.221-248, 1994.



**Shadan Saniepour E.** received BSc. in Electronics and Telecommunication Engineering in 1993 from Iran University of Science and Technology — Iran and MSc. from Saitama University — Japan in 1999. She is currently a Ph.D. candidate at the Department of Information and Computer Sciences, Saitama University. Her research interests are in network and web server traffic engineering and performance management.



**Behrouz Homayoun Far** received BSc. and MSc. degrees in Electronic Engineering in 1983 and 1986, respectively, from Tehran University, Iran. He has received his Ph.D. degree from Chiba University — Japan, in 1990. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, University of Calgary, Canada, where he is the coordinator of the Intelligent Systems Group at The University of Calgary.

The research fields of his interest are automatic programming, software quality management and distributed AI. Dr. Far is a member of the ACM, IEEE Computer society, JSAI, and IPSJ.